L6: Entry 4 of 71 File: PGPB Dec 12, 2002

DOCUMENT-IDENTIFIER: US 20020188666 A1
TITLE: Lightweight dynamic service conversation controller

Application Filing Date:
20010523

Detail Description Paragraph:
[0028] When a conversation proceeds from one state to another, a "state" of the conversation, which contains information of the current state, may need to be tracked. If the conversation controller maintains the state of the conversation, the conversation controller may be referred to as stateful. However, if the "state" of the conversation is carried in the message and passed from the client and the server to the conversation controller, the conversation controller may be referred to as stateless. A stateless conversation controller may be easier to implement, while a stateful conversation controller may be extended to implement performance management, conversation history, or rollback mechanisms, and thus may be more effective in handling issues such as malicious behavior on the part of one of the participants.

Detail Description Paragraph:
[0035] The conversation controller may maintain and track the "state" of the conversation, i.e., implemented as stateful, step 212, or may retrieve the "state" of the conversation from the service, i.e., implemented as stateless, step 214.

Detail Description Paragraph:
[0048] FIG. 3 is a block diagram illustrating the components of a conversation controller 300, from the perspective of how the conversation controller 300 manages messages from a client 360 to a service 370. An incoming context handler 310 may include a logic that manages message structure, and may be responsible for restoring contextual information packed in incoming message headers to its original format. An outgoing content handler 320 maybe responsible for packing contextual information into outgoing message headers and composing outgoing messages. An interaction handler 330 may parse and query conversation definitions to, for example, validate document types or calculate new conversation states. The interaction handler 330 may also raise exceptions when an invalid type document is received. A dispatch handler 340 may parse and query service descriptions, typically specified in WSDL, and use the service descriptions to forward messages to the service 370. An optional client interaction controller handler 350 and interactions, drawn with dotted lines, may dispatch a reply from the service 370 to the client 360 and then forward the client's response back to the service 370 by way of the conversation controller 300.

Previous Doc Next Doc Go to Doc#
First Hit

Generate Collection

L6: Entry 5 of 71 File: PGPB Apr 18, 2002

DOCUMENT-IDENTIFIER: US 20020046230 A1
TITLE: METHOD FOR SCHEDULING THREAD EXECUTION ON A LIMITED NUMBER OF OPERATING SYSTEM THREADS

Application Filing Date:
19980429

Detail Description Paragraph:
[0024] The threading services 14 of the flow scheduler 10 provide for exception handling by allowing each interface object to register an exception handler. The flow scheduler 10 will call the registered routine and pass an indication of the kind of exception to the registered handler when a lower level interface generates an exception. During normal operations, the exception handler is called when the context thread is scheduled to run. An interface, therefore, is not required to account for reentrancy. However, the flow scheduler 10 provides for true asynchronous exception delivery when it is desirable. This may be the case for lower level interfaces that are performing I/O operations that may take some time to complete. An interface must explicitly enable asynchronous exception delivery. In general, for exception handling to work correctly, either all or none of the interfaces should provide for exception handling with the context thread. The flow scheduler 10 will continue normal execution even while an exception is outstanding if the active interface has not provided an exception handler. This may be a problem when a lower level interface generates an exception when its parent does not provide an exception handler. It will continue execution as if the exception did not happen. However, this may be acceptable for timeout and cancel exceptions. The timeout or cancel is handled as soon as possible but may be deferred to a higher level interface better capable of handling it.

Detail Description Paragraph:
[0026] When a context thread is timed out, the active interface's exception handler is called. To prevent excessive synchronization within the threading services 14 and reentrancy control within the applets and elements, the exception routine is not called asynchronously on the interface that is currently executing. The existing execution function is allowed to complete before timeout or cancel is signaled. Although this behavior helps simplify time-out and request cancellation, it may not be desirable when an element is executing a synchronous I/O operation. To help solve this limitation, the threading service 14 provides primitives to place the active ED object into a state which will accept asynchronous exceptions. In this mode, time-out or cancel processing may call the exception routine for the active ED object while the ED object is executing. The threading services 14 will use a lock to prevent damage to internal data structures. The element may also need to provide locks for any data associated with its exception routine, since two execution threads may be executing in the same ED object context concurrently. An asynchronous exception is typically used by elements providing access to I/O devices with an unpredictable latency. When servicing a synchronous request, these elements will block the execution thread awaiting I/O completion. The asynchronous exception will allow the element to cancel the I/O operation which in turn will return the execution thread to the element.

Detail Description Paragraph:
[0038] Applets embody algorithms for performing operations that are stateless, i.e., there are no global variables or data structures available to applets. All data that is processed by the applets are either encapsulated objects or system-owned structures. Any state needed for the life of an applet must be stored in a thread context object.

Detail Description Paragraph:
[0103] The BLOCK statement creates a scope for additional local variables to be declared. Local variables declared in the <local variables> clause are allocated and constructed on entry to the BLOCK. They are destroyed and deallocated on exit from the block statement. The statement implicitly generates an exception handler so that the storage is deallocated even when an exception is thrown from within the block. BLOCK statements can be arbitrarily nested. Variables in an inner scope can have the same names and occlude variables in an outer scope.

Detail Description Paragraph:
[0147] The LOCK statement allows the programming language to provide structured locking. An applet may pend until an acquired lock can be granted. Once the lock is granted, the statement list associated with the LOCK statement is executed. The LOCK-ENDLOCK scope should create an exception handler so that exceptions from within the block will flow to the handler and the lock will be unlocked in all cases on exit from the block. This should be done implicitly.

Detail Description Paragraph:
[0176] The TRY command provides structured exception handling. The programming language should not allow TRY statements can be arbitrarily nested and should not allow a RETURN statement from within a TRY, since a RETURN completes the executing applet and destroys the TRY stack within that execution context. TRY-FINALLY code should not be executed because exception state may be lost.

Detail Description Paragraph:
[0181] The UNLOCK-ENDUNLOCK scope should implicitly create an exception handler so that exceptions from within the block will flow to the handler and the lock will be relocked in all cases on exit from the block.

Previous Doc Next Doc Go to Doc#